

嵌入式 Linux 系统 开发手册

郭万永(yong_bvcom@sina.com.cn)
2003-6-23

目录

引言

第一章 嵌入 linux 简介

- 1. 1 嵌入 linux 发展
- 1. 2 嵌入 linux 开发步骤简介
- 1. 3 嵌入 linux 开发模型

第二章 嵌入 linux 开发前期准备工作

- 2. 1 主机系统
 - 2. 1. 1 安装开发编译环境
 - 2. 1. 2 配置 DHCP
 - 2. 1. 3 配置 TFTP
 - 2. 1. 4 配置 NFS
- 2. 2 目标系统
- 2. 3 软件包获取

第三章 PPCB00T/U-B00T 的移植

- 3. 1 PPCB00T/U-B00T 简介
- 3. 2 U-B00T 源代码分析
 - 3. 2. 1 与目标板相关的代码
 - 3. 2. 2 与 CPU 相关的代码
 - 3. 2. 3 头文件
 - 3. 3. 4 公共代码
 - 3. 3. 5 网络传输代码
 - 3. 3. 6 Makefile 文件
 - 3. 3. 7 关键的驱动程序文件
- 3. 3 移植步骤
 - 3. 3. 1 编译测试
 - 3. 3. 2 U-B00T 的启动过程
 - 3. 3. 3 添加自己的开发配置文件
 - 3. 3. 4 目标板系统参数配置
 - 3. 3. 5 串口初始化
 - 3. 3. 6 SDRAM 初始化
 - 3. 3. 7 FLASH 驱动
 - 3. 3. 8 FEC 以太网口初始化
 - 3. 3. 9 其它
- 3. 4 编译调试
 - 3. 4. 1 编译
 - 3. 4. 2 目标代码烧写
 - 3. 4. 3 调试

3. 5 U-BOOT 命令

3. 6 小结

第四章 嵌入 Linux 的移植

4. 1 整体分析

4. 1. 1 平台支持

4. 1. 2 组件支持

4. 1. 3 文件系统

4. 1. 4 网络支持

4. 2 内核源代码简单分析

4. 3 重要代码说明及修改

4. 3. 1 串口驱动

4. 3. 2 FLASH 驱动及 MTD 支持

4. 3. 3 FEC 以太网口驱动

4. 4 内核编译控制

4. 4. 1 内核编译

4. 4. 2 内核调试选项

4. 5 内核的装载

4. 6 引导嵌入 Linux 系统

4. 7 关于文件系统

4. 7. 1 简介

4. 7. 2 嵌入 Linux 文件系统分析

4. 7. 3 DOC 支持

4. 7. 4 实例分析—在 FLASH 上构造 JFFS2 文件系统

4. 8 小结

第五章 应用程序开发

5. 1 简介

5. 2 实例：TCP SERVER 及 CLIENT

第六章 SNMP 的实现

第七章 WEB 的支持

引言：

本手册结合 MPC860T 硬件平台，详细叙述了 PPCB00T/UB00T 以及嵌入 Linux 的移植过程，介绍了 SNMP 和 WEB 组件在嵌入平台上的移植过程，同时这也可以作为在其它硬件平台上移植嵌入 Linux 的参考手册。

鉴于篇幅有限，本文将不会列出所涉及的源代码，主要目的是讨论进行 PPCB00T/UB00T 和嵌入 Linux 移植的步骤和通常做法，不会对源代码做过多解释叙述。

第一章 嵌入 linux 简介

1.1 嵌入 linux 的发展

伴随着 linux 的快速成长, 嵌入 linux 在近两年也发展迅速, 与传统商业性操作系统象 vxworks、psos 等相比, 嵌入 linux 没有昂贵的版权费, 而且完全开放源代码, 在 internet 上有着丰富的开发资源, 支持众多 CPU 架构象 PPC、COLD FIRE、ARM、X86、MIPS, 这些有点吸引了众多的商家投入 linux 的怀抱。

与 vxworks 等典型嵌入操作系统相比, 实时性是 Linux 的弱项, 因此一些公司对 linux 进行了二次包装, 以提高嵌入 linux 的实时性, 比较优秀的嵌入 linux 有 Montavista 公司的 Hardhat Linux, 还有 RTLinux、Bluecat Linux 等。

优秀的网络性能是 linux 的招牌, 嵌入 linux 同样对网络有着良好的支持, 支持 TCP/IP、PPP、IGMP、IPV6, 而且提供了对 ATM 协议的支持。

1.2 嵌入 linux 的开发步骤

嵌入 linux 的移植开发是一项很有趣的工作, 在开发的过程中会感受到 linux 强大的吸引力。

首先我们以我们常见的 PC 来描述一下整个系统的工作流程。从打开电源的一刹那, 你的 CPU 就开始了第一次心跳 (上电复位), 然后 CPU 会迈出自己的第一步——跳向指令的入口地址, 通常是在 EEPROM, FLASH ROM 等永久记忆体当中, 然后 CPU 就会按部就班的进行对自身进行最初的初始化工作, 通常这些工作都是由汇编代码完成的, 因为有一些寄存器只有用汇编语言才可以进行读写。

一旦跳到 C 语言的领域, 一切变的明朗起来, 无非就是对外围设备的初始化, 对 DRAM/SDRAM 的初始化, 对串口、网口等初始化, 对总线设备的初始化等, 不过虽然说起来容易, 但事实上非常复杂, 因为你要对你的所有硬件设备都很了解, 这些工作完成之后就要将接力棒传给 Linux 的内核了, 上面的工作就是我们常说的计算机 BIOS 所要做的工作, 在嵌入 Linux 系统中我们给它取了新的名字, BSP (board support package) 板级支持包, Bootloader 引导者。

BIOS 的工作完成之后就会将 CPU 完整的交给 Linux kernel, 此时 BIOS 的使命就结束了, 剩下的就是 linux 内核启动, 加载设备驱动程序, 初始化网络协议, 启动系统服务象 FTP, WEB, SNMP 等, 详细的过程在随后的章节中在叙述。

怎么样, 我想现在大概有个简单的概念了吧! 嵌入 linux 系统从在这里我们暂且把它分为三个大部分: 硬件系统、bootloader 和 linux kernel, 他们的关系如下图:

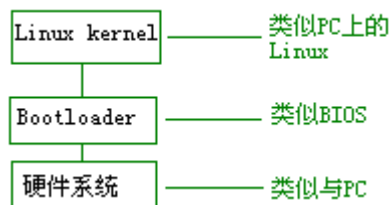


图 1 嵌入 linux 系统概念图

按照这个概念图来说, 我们的开发步骤很简单:

- 设计自己的硬件系统

- 编写 boot loader
- 裁减自己的 linux 内核

是不是太简单了，别着急，后面几章会一一把他们分解开来详细叙述。

1.3 嵌入 linux 开发模型

通常嵌入 linux 的开发分为主机系统和目标系统，他们之间通过网络接口和串口互连，网口一般用以下载程序和内核映像文件，串口一般作为 console 控制台来使用，在主机通过超级终端与目标系统进行命令交互，就像 PC 的键盘和显示器。

如果你第一次接触嵌入系统，可能会问什么是主机系统和目标系统？很简单，主机系统就是我们一般用的计算机或者工作站，用来编写代码和编译；目标系统就是你的嵌入开发系统中的硬件板。在主机开发好的目标执行代码通过网络接口或者烧写到目标系统中的永久记忆体（通常是 flash 或者 DOC -Disk On Chip），目标系统执行这些代码。

模型如下图所示：

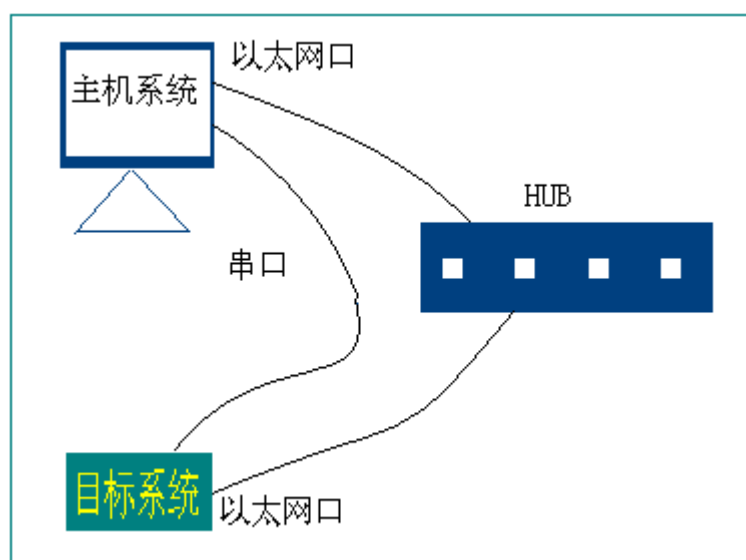


图 2 嵌入 linux 系统开发模型

很简单吧，不过要注意，虽然需求很少，但有两个必要：主机至少要有个 232 的通用串口（有些笔记本上却没有），还要有一个 10M 或 100M 的以太网口（关键看你的目标系统是那种网络接口），然后还有一根串口线和两根网线，如果不用 HUB 的话，一根单机对联线就解决问题了。硬件系统这样就搞定了！

第二章 嵌入 linux 开发前期准备工作

硬件系统搞定之后，还有一些软件的准备工作，俗话说“磨刀不误砍柴功！”，在未进行正式的开发工作之前，对系统有个全面的认识，做好充足的准备，会避免好多后期将可能会出现的问题。

2. 1 主机系统

前面我们讲过，主机系统通常就是我们使用的计算机或工作站，当然操作系统要是 Linux，这里我的主机系统为硬件配置为：

CPU：PIII550

SDRAM：256MB

HARDDISK：20G

100M 快速以太网卡一块，通用串行 232 接口

操作系统：

Redhat Linux8.0 完全安装

2. 1. 1 安装开发编译环境

主机准备好之后还要安装交叉编译环境，用于编译目标系统可执行的代码。为什么要这个交叉编译环境呢？大家知道一般我们用的 PC CPU 的架构为 X86，我们的目标系统为 8XX 架构的，所以要有一个编译器，链接器用于将 C 语言的代码编译为 8XX 系统可识别的目标代码。

你完全可以自己建立嵌入 linux 的交叉编译环境，可是有点烦琐，所以如果你有 Montavista 的 CDK，那么是最好的选择了，如果没有，也可以到德国的 DENX 软件中心的 FTP 站点上下载他们的 ELDK。这里以 CDK 为例，安装很简单，执行一个简单的 Install 即可，安装完成之后会在一个固定的目录下多了一些可执行文件，以 8xx 系统为例，在*/bin/下会有 ppc_8xx-gcc ppc_8xx-objcopy ppc_8xx-ld 等。

ELDK 的安装相对也很简单，下载它的 ELDK，实际是一个编译控制文件 Makefile，你要将所需要的源代码（一般为 tar 格式的）下载之后放到它指定的目录中，然后执行 make，如果没有什么错误，就会在根目录下产生一个新目录 LinuxPPC，里面就是产生的编译器连接器的执行文件。这里不再详细叙述了，相关的说明可以在他们的 Readme 中看到。

安装好之后可以用“which ppc_8xx-gcc”来查找一下产生的目标执行文件在哪里，如果告诉你没有发现，那么很不幸你的安装没有成功（当然，首先你要将路径加到系统路径中）。

2. 1. 2 配置 DHCP

主机系统需要设置 DHCP 服务器，用于目标系统在采用 BOOTP 启动时获取 IP 地址。设置之前首先要知道目标系统的 MAC 地址，同时要保证主机系统安装了 DHCPD 服务器。下面是一个典型配置，在/etc/dhcpd.conf 文件中：

```
ddns-update-style none;
allow bootp;
subnet 200.162.162.0 netmask 255.255.255.0 {
    default-lease-time 1209600;
    max-lease-time 31557600;
group {
```

```
host linux-8xx {  
hardware Ethernet 08:00:3e:26:9d:5b;  
fixed-address 200.162.162.190;  
option root-path "/dev/nfs";  
}  
}  
}
```

用如下命令重新启动 dhcpd 服务：

```
#/etc/init.d/dhcpd restart
```

同时启动 dhcpd 服务，使主机每次启动时自动启动该服务。

2. 1. 3 配置 TFTP 服务器

TFTP（简单文件传输协议）服务器用于目标系统从主机系统上获取可执行的代码或内核映像文件。

以超级用户登录主机系统，编辑下面文件：

```
#vim /etc/xinetd.conf
```

去掉下面一行的注释：

```
tftp dgram udp wait root /usr/sbin/tcftp in.tftpd
```

根目录下创建 tftpboot 目录：

```
#mkdir tftpboot
```

将每次编译好的目标代码拷贝到该目录下，在系统服务中启动 TFTP 服务。

（运行#setup，在“System services”中选择。）然后重新启动系统即可。

2. 1. 4 配置 NFS

NFS 在我们的开发系统中主要是给目标系统提供一个根文件系统（Root Filesystem）。

首先，编辑配置文件/etc/exports：

```
#cd /etc
```

```
#vim exports
```

如果原来不存在，添加一行：

```
/opt/nfsfilesystem/ *(rw,insecure, sync,no_root_squash)
```

“/opt/nfsfilesystem/”是根文件系统所在的目录，“*”允许主机所在网段所有机器访问。

和配置 TFTP 一样，在系统服务中启动 nfs 服务，重新启动系统或运行：

```
#/etc/init.d/nfs restart
```

至此，主机配置基本完毕，前期的准备工作基本搞定，还有一些小工作要做！

2. 2 目标系统

目标系统就是我们的开发板，我们用的是自己开发的 MPC860T 开发板的一个最小系统。一般的最小系统要包括以下几个方面：

CPU 肯定是要有的了；

SDRAM/DRAM 最好 32M 或更大（反正这东西便宜）；

FLASH 或者 DOC：FLASH 最好 8M 或更大，这样可以建立一个可读写的文件系统，DOC 最大可以到 100 多 MB，也是不错的选择。

以太网口，当然你不要也可以，从串口下载程序吧！

串口，这可是必须的！

我们的目标系统如下：

CPU：MPC860T 80MHz

SDRAM：32MB

FLASH：8/16MB

网口：FEC 100Mb

串口：rs232 X 2

目标系统硬件没什么可以说的，很简单！这里提一下一个比较重要的问题，要做好整个系统的地址分配，这在后面会用到！

2.3 软件包获取

最后要获取相关的软件开发包，假设你什么也没有，那么去 DENX 的 FTP 看看，基本可以满足需要了：<ftp.denx.de>。

主要获取以下软件包：

某个版本的 PPCBOOT/UBOOT 源代码

ELDK（如果你有 CDK，这就不用了）

某个版本的 LINUX KERNEL

下载这些软件包之后在某个目录下解压，以我们的为例：

下载 u-boot0.2.0.tar.bz2 到目录/opt；

解压：

```
#cd /opt
```

```
#bzip2 -cd u-boot0.2.0 | tar xf -
```

解压完成之后会产生一个目录“u-boot-0.2.0”。

内核的解压和这差不多。

当然，在 Internet 上有很多可以获取这些代码的地方，这里不一一列举了，举几个网站的例子：

<http://penguinppc.org/>

<http://qslinux.org/>

<http://lists.linuxppc.org/>

www.hardhat.com

<http://ppcboot.sourceforge.net/>

第三章 PPCBOOT/U-BOOT 的移植

3.1 PPCBOOT/U-BOOT 简介

ppcboot 的前身是 8XXROM 和 FADSR0M, 其最高版本为 PPCBOOT2.0, 2.0 版本以后名称就改为了 U-BOOT, 是由德国 DENX 软件中心依照 GPL 发布的公共软件, 本手册我们结合所用的 u-boot-0.2.0 (其版本划分为 u-boot-*.*, 分别为 version.subversion.patchlevel) 来详细叙述整个的 porting 过程。我们所用的版本为 u-boot-0.2.0, 支持 BOOTP, TFTP, SERIAL 代码下载, 后续版本可以引导 OpenBSD, NetBSD, FreeBSD, 4.4BSD, Linux, SVR4, Esix, Solaris, Irix, SCO, Dell, NCR, VxWorks, LynxOS, pSOS, QNX 内核映象, 目前支持 NetBSD, Linux, Vxworks 和 QNX; 支持的目标 CPU 架构 Alpha, ARM, Intel x86, IA64, MIPS, MIPS, PowerPC, IBM S390, SuperH, Sparc, Sparc 64 Bit, 目前支持 PowerPC。详细的说明可以参见 U-BOOT 目录下的 README 文件。

3.2 U-BOOT 源代码分析

经过漫长的准备工作, 现在终于可以进行实质性的工作了! 首先我们来分析一下 U-BOOT 的源代码分布, 这样才可以做到“知己知彼, 百战不殆”! 首先列出它的源码树 (图 3), 然后再解剖它。

3.2.1 与目标板相关的代码

目前 U-BOOT 支持大多数比较常见的目标板, 象 MOTOROLA 的 ADS 和 FADS 板, 一般我们设计的主板和这些主板都是大同小异, 不会有太大的差别。源代码树中 board/下的每个文件夹对应一个或几个主板, 象/board/fads 目录对应的就是 FADS 板和 ADS 板的源代码, 一些主板上的资源例如 SDRAM 和 FLASH 的初始化代码就在这个文件夹中。

3.2.2 与 CPU 相关的代码

该版本的 U-BOOT 只支持 PowerPC 系列的主板, 与 CPU 相关的代码未于源代码树下的 CPU/下的各个目录当中, 例如我们用的相关 CPU 的代码就位于/cpu/mpc8xx/文件夹中, 其中 start.S 是 U-BOOT 的入口代码, 用汇编语言写的, 是整个代码的入口, serial.c 是串口的初始化代码。相应的, 目录/cpu/mpc8260/对应的就是 8260CPU 的相关代码。

3.2.3 头文件

头文件位于源码树下的 include/下, 其中各种主板的配置文件位于/include/configs/文件夹中, 象 FADS860T 主板的配置文件为/include/configs/FADS860T.h; 目录/include/asm-ppc 是一些比较底层的头文件, 编译时会根据不同的配置与/include/configs/asm 建立一个符号链接。

3.2.4 公共代码

除了和主板, CPU 特性相关的代码外, 其他大部分都是公用的代码, 位于/common/下面, 象 U-BOOT 的命令解析代码/common/command.c, U-BOOT 环境变量处理代码 environment.c 等都位于该目录下。还有一个公共的代码就是 PowerPC 系列的主板初始化代码也是公用的, 位于/lib_ppc/board.c 中, 在完成 CPU 的初始化之后就会跳到这个文件中, 而与主板相关代码的执行都是在这个文件中调用的。

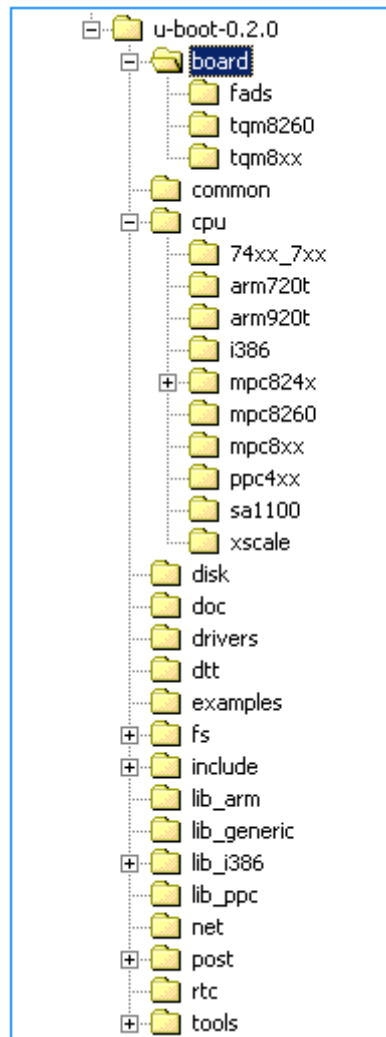


图 3 U-B00T 源代码树

3. 3. 5 网络传输代码

网络传输代码位于目录中的/net/下面，象 arp.c , bootp.c, eth.c, tftp.c 等都在这里，B00TP, TFTP，以太网初始化等功能的实现也是由这些代码完成的

3. 3. 6 Makefile 文件

Makefile 文件位于 U-B00T 的根目录下，是整个编译的控制主文件，后面我们会讲到如何修改该文件以添加我们自己主板的编译控制。

3. 3. 7 关键的驱动程序文件

U-B00T 中几个比较关键的外设驱动代码包括串口的初始化，以太网口驱动，SDRAM 初始化，FLASH 初始化，以 FADS 板为例，分别位于：

串口： /cpu/mpc8xx/serial.c

SDRAM: /board/fads/fads.c

FLASH: /board/fads/flash.c

ETHERNET: /cpu/mpc8xx/scc.c

FAST ETHERNET: /cpu/mpc8xx/fec.c

在我们移植的过程中这些都是我们要修改和关注的代码。

3. 3 移植步骤

好了，对 U-B00T 的代码有了了解之后，下面我们就一步一步的在做一个可以在我们自己主板上跑的 U-B00T！

3. 3. 1 编译测试

在主机开发系统上进入 U-B00T 的目录中，以我的系统为例（后面的命令皆以我的系统为例）：`#cd /opt/u-boot-0.2.0`，在进行编译之前设置目标板配置，U-B00T 支持的主板种类在其目录中的 README 中可以查到，针对 FADS860T 主板：

`#make FADS860T_config` 屏幕出现：

```
rm -f include/config.h include/config.mk
```

Configuring for FADS860T board...

配置成功，然后运行命令：

```
#make
```

进行编译，编译完毕产生目标代码 `u-boot.srec` `u-boot.bin` `u-boot`，说明 U-B00T 和交叉编译环境都可以正常使用，下面我们就可以放心的做我们自己的主板 U-B00T 了，如果有错误出现，至少不用怀疑 U-B00T 和交叉编译环境了（不是完完全全不用啊！！）。

3. 3. 2 U-B00T 启动过程

从哪里入手呢？先看看它的启动过程待会就知道从哪里入手了！

CPU 入口函数（FLASH 中运行）： `/cpu/mpc8xx/start.S`

CPU 初始化（FLASH 中运行）： `/cpu/mpc8xx/cpu_init.c`

主板初始化（FLASH 中运行）： `/lib_ppc/board.c`（调用串口初始化：`/cpu/mpc8xx/serial.c`，sdrain 初始化：`/board/fads/fads.c`）

主板初始化（RAM 中运行）： `/lib_ppc/board.c`（调用其它公共函数）

3. 3. 3 添加自己的配置文件

参照 Makefile 文件，选择一个与自己目标板最为相近的一个主板，我们选择 FADS860T 板，在 Makefile 文件中

```
FADS860T_config:    unconfig

    @./mkconfig $(@:_config=) ppc mpc8xx fads
```

后面添加：

```
SUMMER860T_config: unconfig

    @./mkconfig $(@:_config=) ppc mpc8xx fads
```

其中 ppc 是 CPU 的架构， mpc8xx 是 CPU 对应的目录， fads 是主板对应的目录，也可以新建一个目录，然后指向新的目录。

Makefile 修改完毕还要添加对应的头文件，SUMMER860T_config 对应的头文件应该是 SUMMER860T.h，在 /include/configs/ 下添加该文件，你可以将 FADS860T 拷贝一份，命名为 SUMMER860T.h：

```
#cd /opt/u-boot-0.2.0/include/configs

#cp FADS860T.h SUMMER860T.h
```

完成之后，可以测试一下你的配置：

```
#make SUMMER860T_config

#make
```

编译成功！OK 了，当然这时你编译出来的目标代码和 FADS860T 是一模一样的，下一步我们要根据我们的目标板进行修改了！

3. 3. 4 目标板系统参数配置

目标板系统参数在 /include/configs/SUMMER860T.h 中配置，实际只要我们对对自己主板有足够的了解，然后对 FADS 板有足够了解，然后修改一二即可以了，下面是我修改的参数以及一些比较关键的配置说明：

```
#define CONFIG_MPC860    1
```

这不用说了，肯定是必须的

```
#define CONFIG_MPC860T    1
```

MPC860 的家族有好几种 CPU，他们之间有一些差别，如果你的 CPU 是 MPC860T，这个可别忘了，不然到时你的 FEC 至少会跑不起来的，其它的我还不知道！

```
#define CONFIG_FADS      1
```

FADS 用的是 SDRAM，如果你比较懒，把这个也定义上，到时把相关代码和你主板对应起来也可以用了

```
#define CONFIG_8xx_CONS_SMC1  1
```

用 SMC1 当串口使用，作为你的 CONSOLE

```
#define CONFIG_BAUDRATE      19200
```

串口的波特率，到时你的主机系统的超级终端要和这个对应，不然超级终端会有一堆乱码出现的

```
#define MPC8XX_FACT      1
```

晶阵倍频，FADS 是 12，我的主板设为 1

```
#define MPC8XX_XIN          50000000          /* 4 MHz in */
```

晶阵频率，FADS860T 是 4MHz 的，我的主板是 50MHz 的

```
#define MPC8XX_HZ ((MPC8XX_XIN) * (MPC8XX_FACT))
```

工作频率

```
#undef  CONFIG_WATCHDOG          /* watchdog disabled */
```

WATCHDOG 先关掉

```
#define CFG_SDRAM_BASE      0x00000000
```

SDRAM 起始地址，这可不能错！

```
#define CFG_FLASH_BASE      0x02800000
```

FLASH 的起始地址，我为它分配的基地址为 0x2800000

```
#define CFG_FLASH_SIZE      ((uint)(2 * 1024 * 1024)) /* max 8Mbyte */
```

FLASH 的大小，悲惨，我只有 2MB，不过先凑合着用，最后我会有 8MB，到时又得修改它了

```
#define CFG_ENV_IS_IN_FLASH 1
```

U-BOOT 的环境变量参数在 FLASH 中，没错，我的就是在 FLASH 中

```
#define CFG_ENV_OFFSET      0x1fc000
```

FLASH 有 TOP-BOOT-SECTOR 和 BOTTOM-BOOT-SECTOR 之分（不知是否有那种每个 SECTOR 都一般大的，应该有吧？）我们用的是 TOP-BOOT-SECTOR，我把 U-BOOT 的环境变量存在 FLASH 的最后一个 SECTOR 中，这是它在 FLASH 中的偏移地址。

```
#define CFG_ENV_SIZE        0x4000 /* Total Size of Environment Sector */
```

U-BOOT 环境变量所占空间大小，这个足够了

```
#define CFG_ENV_SECT_SIZE   0x4000 /* see README - env sector total size */
```

U-BOOT 环境变量所在 FLASH SECTOR 的空间大小，README 中说要修改 u-boot.lds，可是修改时有错误，干脆不修改了，也可以使用，是不是有什么潜在的危险呀？

```
#define CFG_IMMR            0xff000000
```

随便找个你没有用到的地址空间分配给它吧，它是内部寄存器的基地址，我给他分配我 0xff000000，不过要保证这个基地址要有 64K 大小的访问空间，这个地址范围可不能和其它外设的地址范围有重叠

下面还有一些寄存器的初始值设置，我没有做什么修改，但和 MEMORYCONTROLLER 有关的寄存器还是有修改的，根据自己的主板特性，修改吧！涉及以下寄存器：

OR0—OR7 BR0-BR7 MAMR/MBMR 等。

其中，OR BR 寄存器的设置是对应你目标系统的地址划分的，你可以灵活的编程来确定每一个片选 CS0—CS7 所对应的地址空间。

目标板配置文件修改的差不多了，下面要进行代码修改了！

3.3.5 串口初始化

CPU 的入口函数/cpu/mpc8xx/start.S 不用做修改（反正我的主板没有做修改），在进行完 CPU 的上电初始化之后就跳入/cpu/mpc8xx/cpu_init.c 进行一些寄存器的初始化，讲起来太费时间了，找个 860UM 看看吧。这里我基本没有修改。CPU 初始化完成后就调用/lib_ppc/board.c 中的 board_init_r()，首先就进行串口初始化，因为串口正常工作之后我们就可以从超级终端上看到调试信息了，那就方便了。

串口初始化代码位于/cpu/mpc8xx/serial.c，FADS 板上有一个 CPLD 的逻辑，我们的目标板上没有，所以对对应于 BCSR 的设置改为自己相应管脚配置。其它不用做什么修改。

这里顺便提一下，串口初始化完成之后要调用 check_cpu()，check_board()，在 check_board() 中 FADS 板会有一些访问 BCSR 地址的代码，我把他们屏蔽掉了！

3.3.6 SDRAM 初始化

上面的工作全部是在 FLASH 中进行的，程序的限制很多，U-BOOT 仅仅是在 MPC860T 内部的一小片 RAM 中开了一个小小的堆栈用于程序使用，你不能定义大的变量，郁闷！好，马上进行我们 32MB SDRAM 的初始化，完成后就可以为所欲为了。

SDRAM 的初始化代码位于/board/fads/fads.c 中，对应不同的 SDRAM，它的 UPM 表是不一样的。什么是 UPM 表？UPM 表说白了就是一些总线时序抽象为十六进制数字的产物，用数字来表示 SDRAM 的读写，刷新等时序。我是问我们硬件工程师要的，密密麻麻一堆数字。还有一个 SDRAM 上电的初始化时序，叫 SDRAM POWERUP UPM 表。修改时要注意，这里就不列出修改后的代码了。

SDRAM 的初始化是/lib_ppc/board.c 中的 board_init_f()调用的，入口为 intidram()。

3.3.7 FLASH 驱动

在进行完 SDRAM 的初始化之后，U-BOOT 会将代码拷贝到 SDRAM 中，在 SDRAM 中再次对 CPU 和 CONSOLE 进行一次初始化（参阅函数/lib_ppc/board.c 中的 board_init_r()），这些代码都基本不用做修改，其中对 FLASH 的初始化是我们要修改的一个部分。

FLASH 的驱动位于/board/fads/flash.c，代码中涉及 FLASH 型号的探测，擦除，编程，扇区保护和取消保护等操作。其中有一个公共结构体 flash_info_t flash_info[CFG_MAX_FLASH_BANKS]用于保存 FLASH 的参数。这部分要根据目标系统所用的 FLASH 的型号进行修改，要注意的是 FLASH 的数据总线宽度，是 8bit,16bit 还是 32bit，其它按照 FLASH 的编程手册修改驱动就可以了。

3.3.8 FEC 快速以太网口初始化

FEC 快速以太网口的初始化也是在/lib_ppc/board.c 中的 board_init_r()函数中完成的，调用 eth_initialize()，该函数位于/net/eth.c 中，其中调用 fec_initialize()注册了快速以太网设备 FEC。

FEC 的驱动程序位于/cpu/mpc8xx/fec.c 文件中，我们目标系统用的 PHY 为 LX970A，不用做大的修改，只要将 FADS 中涉及到 BCSR 控制管脚的地方改为目标系统想对应的管脚就可以了。详细的代码请参阅/cpu/mpc8xx/fec.c。

3.3.9 其它

至此，代码修改基本完成了。其它的比如打开调试信息在/include/common.h 中定义“#define DEBUG”，此外你也可以在你比较关注的地方自己添加调试信息。

3.4 编译调试

3.4.1 编译

代码修改完毕，进行编译吧！编译之前要根据你目标系统程序入口修改你的程序代码地址（位于/board/fads/config.mk 中），我们的目标系统和 FADS 一样为 0x2800000，没有进行修改。然后运行前面提到的命令（当然要在 U-B00T 的目录下）：

```
#make SUMMER860T_config
```

```
#make
```

当然，如果你比较粗心，会出来一堆错误，不过没关系，一个一个的消灭，可以把编译信息输出到一个文件中，这样比较方便查找问题所在，用下面指令替代“#make”：

```
#make &>out
```

然后可以用：

```
#more out | less
```

 查看你哪里犯了错误，把他们搞定之后，编译产生了目标执行代码 u-boot.srec, u-boot.bin, u-boot，编译完成了！

3. 4. 2 目标代码烧写

对于生成的三个目标文件，u-boot.srec 是 MOTOROLA S-Recoder 格式的，u-boot.bin 是二进制的，u-boot 是 ELF（Executable and Link Format）二进制格式的。

DENX 的网站上说他们烧写 FLASH 用的是硬件 BDM 仿真器 BDI2000，对这个东西不是很了解，我们用的是 WindRiver 的 vision probe BDM 仿真器，当然你用其它的也可以，只要能将代码烧写到 FLASH 中就行。我用的是 ELF 二进制文件，用 BDM 调试仿真环境 vision click7.8 将 u-boot 进行转换获得 u-boot.bin，然后烧写到 FLASH 中，记得在 FLASH 的偏移地址要为 0x100（针对 ppc 系列），烧写 OK！

3. 4. 3 调试

好了，所有的工作基本都完成了，现在激动人心的时刻来临了，关闭目标系统电源，拔掉 BDM 仿真器，主机系统打开超级终端，目标系统加电，启动…，超级终端没有任何信息出来，失败了！难道不成功？再插上 BDM 仿真器，单步跟踪，将 PC 指针设为 0x2800100，执行，没有问题，程序再执行，说明生成的目标代码是有效的，可能是程序代码有问题！

调试吧，在我们移植的过程中，用 BDM 仿真器无法进行 C 语言的跟踪仿真（在调试 vxworks 时是可以的，可能是某个地方没有做好），另想办法吧！焊几个发光二极管到目标板上，在程序中分段点亮这几个二极管，这样可以看到程序在哪个段落出问题然后再分段查找！还有一个方法就是在程序中你认为可能出问题的地方添加对某个通用寄存器赋值的代码，然后用 vision click 跟踪汇编代码，到出错时查看该通用寄存器，同样可以知道程序大约运行到哪个地方！

好，先焊两个发光二极管（记着用那些没有使用的通用 I/O 口来控制），然后在程序中添加几个赋值语句，为 OR1（我们的开发板没有用到 CS1，如果你的用到了，可不能随便乱改，选一个没有用到的寄存器）赋值，不是没有信息出来吗，首先考虑是不是串口初始化失败，所以在串口初始化中添加赋值语句，重新编译 U-B00T，烧写，重新启动，二极管是亮

的，程序确实在运行，然后就是用 vision click 来跟踪吧，反正我最后找到的问题是在全局变量 gd(global data)结构中有个 gd->flags，在 FLASH 中运行代码和在 RAM 中运行代码时 CONSOLE 是不一样的，但我的 console.c 中在 FLASH 中运行 puts 时确调用的在 RAM 中运行的 fputs，当然会有问题！问题就出在 gd->flags，于是我把它先初始化为 0（在 U-B00T 中没有对 gd 进行初始化，这样是不是会有一些未知的问题出现？至少我觉得 flags 就是一个，或者是我在做 CPU 初始化的时候在别的地方对其进行初始化，因为此时 gd 是在 860 的内部 RAM 中的），改掉之后重新编译，烧写，重新启动，令人振奋 !!! 信息出现了：

U-Boot 0.2.0 (Jun 20 2003 - 15:18:50)

CPU: XPC860xxZPnnD4 at 50 MHz: 4 kB I-Cache 4 kB D-Cache FEC present

Board: SNOIC MPC860T 100M ethernet network board(Ver 1.0).

DRAM: (32 MB SDRAM) 32 MB

FLASH: 2 MB

In: serial

Out: serial

Err: serial

Net: FEC ETHERNET

Hit any key to stop autoboot: 2

当然了，还有其它一些调试过程就不一一叙述了，因为不同的目标系统可能遇到的问题不一样，我在调试的过程中 SDRAM 和 FLASH 都出现或多或少的的问题，但是一旦 console 可以输出调试信息了，所有的问题都变的简单了，你可以随便添加 printf ()，查看任何你想看的信息，这样还有什么搞不定呢？我把调试的过程分为几个阶段：

程序代码可以在目标系统上运行	最基本的
console 可以输出调试信息	具有划时代意义的
可以在 SDRAM 中运行了	胜利的冲锋号
其它	打扫战场

此时，u-boot 的移植工作已经基本完成了。

3. 5 U-B00T 命令

U-B00T 运行稳定后，可以用它的内部命令来查看目标系统的信息，设置环境变量等，

这里就几个比较常用的命令做简单叙述，详细说明在 U-BOOT 目录下的 README 文件中。

- help / ? 帮助命令

该命令用于查询 U-BOOT 支持的命令，并列出简单说明，help 和 ? 是同一命令。

- bdinfo 查看目标系统参数和变量

查看目标板的硬件配置，各种变量参数

- setenv 设置环境变量

比较常用的有

```
setenv ipaddr *.*.*.*
```

```
setenv serverip *.*.*.*
```

```
setenv gatewayip *.*.*.*
```

```
setenv ethaddr *.*.*.*.*.*
```

等

- printenv 查看环境变量

- saveenv 保存设置的环境变量到 FLASH

- mw md mm 一内存的写，查看，修改

- flinfo 查看 flash 的信息

- erase 【startaddr endaddr】 擦除 flash，只能进行一个或多个完整 sector 的擦

- cp 【sourceaddr targetaddr size】 内存复制，可以在 RAM 和 FLASH 中交换数据

- imi 【startaddr】 查看内核映像文件

- bootm 【startaddr】 从某个地址启动内核

- tftpboot 【startaddr imagename】 通过 tftp 从主机系统下载内核映像文件

- reset 系统复位

整个 u-boot 的 porting 过程基本就是这样，可能不同的目标系统会出现不同的问题，这里要将所有遇到的问题都涵盖也是不太现实的，但只要抓住 porting 过程的主线，分阶段将各个问题一一解决，还是比较容易完成的。下面是我工作的一个简单过程，也是 U-BOOT 的推荐过程：

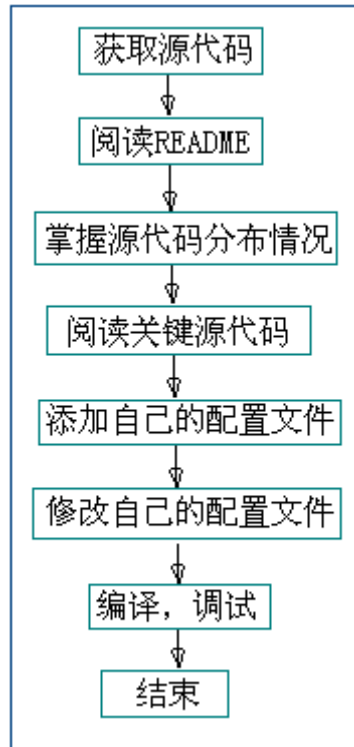


图 4 U-BOOT porting 流程简图

在工作过程中参照了 internet 上的大量文档和说明，这对移植过程有很大的帮助，其中 Linux for PowerPC Embedded System HOWTO 是一个很不错的入门文档：

<http://penguinppc.org/embedded/howto/PowerPC-Embedded-HOWTO.html>

第四章 嵌入式 Linux 的移植

U-BOOT 的工作暂时告一个段落,从本章开始我们将就如何制作自己的嵌入 linux 内核展开讨论。

4. 1 整体分析

Linux 的版本更新很快,我们用的嵌入 Linux 是在 DENX 软件中心的嵌入 Linux 2.4.4,可以从它的 ftp 网站上下载。

4. 4. 1 平台支持

众所周知,linux 是一个可以在众多平台上运行的操作系统,除了支持最为常见的 X86 架构硬件平台以外还支持 arm, alpha, ia64, parisc, mips, mips64, m68k, powerpc 等。但要搞清一个概念,并不是任何你拿到的内核源代码都支持我们的 PPC 的,首先你要确认你拿到的内核已经包含了对 PPC 的支持,否则你要安装 powerpc 的 patch 之后才可以的。因为除了比较上层的模块不具平台相关性之外,有一些比较底层的,比如定时器,中断,CACHE 管理,MMU 等是和硬件紧密相关的,这些部分都是要相关平台的底层代码支持的。

4. 4. 2 组件支持

Linux 的内核版本更新速度很快,每一版本内核的组件支持不尽相同,选一个比较稳定的而且又能满足你要求的内核即可,不用盲目追求版本的最高。

嵌入 Linux 组件支持对我们比较重要的包括串口,快速以太网口,MTD,flash 的支持,对于 flash 的支持,也是在不断的进行更新。

4. 4. 3 文件系统

嵌入 linux 支持很多种文件系统 (filesystem),象平常 windows 使用的 DOS FAT 文件系统,其它的有 ntfs, romfs, cramfs, jffs, jffs2 等。本手册种我们仅仅讨论 romfs, cramfs 和 jffs。

4. 4. 4 网络支持

嵌入 linux 提供众多网络协议支持,TCP/IP,IGMP,PPP,IPX,Appletalk,ATM,IPV6 等。对于网络接口的支持也很多,我们用的快速以太网口的 PHY 为 LX970,在内核中提供了对它的支持。

Linux 的分析不是本手册的主要内容,这里只是就我们比较关心的部分做简单叙述,详细的情况请查阅相关的 linux 内核分析资料和书籍。

4.2 核源代码布局

虽然本文的主要目的不是对 linux 进行详细讨论,但了解 linux 代码的整体布局对我们来说还是有必要的,和 U-BOOT 一样,首先我们列出 linux 内核的源代码树。

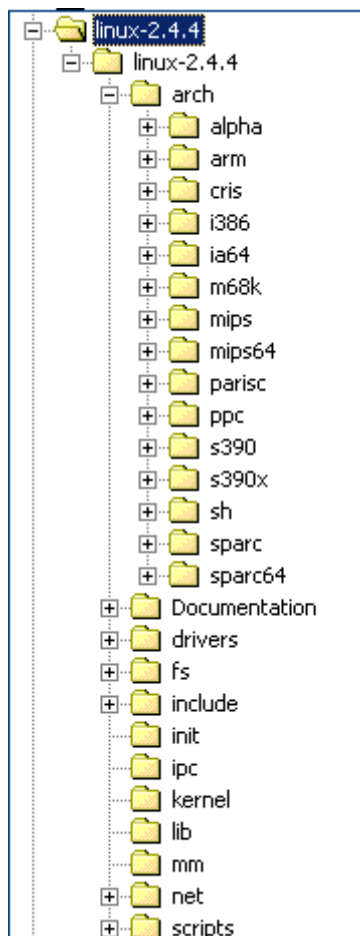


图 5 Linux 内核源代码树

/arch

其中的子目录对应着各种架构 cpu 所特有的代码。

/Documentation

文档

/drivers

该目录包含一个 Makefile 文件和若干子目录,每个子目录对应各种类型的设备驱动程序,象/drivers/char 包含的是字符型的驱动程序,/drivers/block 对应的是一些块设备驱动程序,/driver/net 包含一些网络接口适配器的驱动程序,等等。

/fs

文件处理处于任何 Unix 系统的核心地位,Linux 中的 fs 目录是所有目录中最大的一个。其中包含了当前 Linux 版本所支持的所有文件系统,每个文件系统位于各自的子目录中,还包含了除 fork 和 exit 之外的最重要的系统调用实现。

/include

头文件

/init

包含 linux 的入口函数 main.c，阅读代码可以从这里开始。

/ipc

包含 System V 进程间通信原语的实现代码，分别是信号量、消息队列、共享内存。

/lib

通用库函数，是一个很小的子集，包含了一些通用的支持函数，该目录中最重要的文件是 vsprintf.c，它实现了 vsprintf 函数，而这个函数是 printf 和 printk 的核心。

/mm

该目录包含有内存管理代码。目录中的文件实现了系统中所有与内存管理有关系的数据结构。内存管理和 CPU 架构的特定功能和寄存器是紧密相关的。

/net

该目录包含了套节字（socket）抽象和网络协议的实现。这些功能涉及到大量的代码，因为 linux 支持多种不同的网络协议。每个协议象 IP、IPX 等的实现都位于自己的子目录中。

/scripts

该目录下没有内核源代码，主要包含对内核进行配置的脚本文件。当你运行 make menuconfig 时就是和该目录下的文件进行交互的。

4. 3 代码说明及修改

Linux 的设备驱动程序比较多，我们只就我们比较关心的代码进行简单介绍。

4. 3. 1 串口驱动

串口在嵌入 linux 中是作为控制台使用的，对于 MPC860 来说，SMC1/2, SCC1-4 都可以作为串口使用，我的系统用 SMC1 作为 console 使用，其驱动程序位于 /arch/ppc/8xx_io/uart.c 中，我没有做过任何修改。

4. 3. 2 FLASH 驱动及 MTD 支持

对于嵌入式系统，FLASH 作为内核和文件系统的载体，如果想作为文件系统的载体，必须有 FLASH 的驱动程序，我们用的 FLASH 为 FUJISTU MBM29LV160T，在该内核版本中提供了对这种 FLASH 的支持，如果你用的 FLASH 在该版本中不支持，可以到

<http://www.linux-mtd.infradead.org/>

去下载最新的补丁，以支持更多型号的 FLASH，各种型号 flash 驱动程序位于 /drivers/mtd/chips。

MTD (Memory Technology Device) 是为 linux 开发的中间层软件包，用于将上层的文件系统和底层的存储体（特别是 FLASH）提供一个统一的接口，上层文件系统只是进行读和写操作，具体涉及到硬件的读写则是由 MTD 来完成的，详细情况请参阅：

<http://www.linux-mtd.infradead.org/>

MTD 的驱动程序位于 /drivers/mtd/子目录下。

MTD 功能众多,我也是所知皮毛,仅仅就本系统用到的予以解释。MTD 提供了一个对 FLASH 进行分区的功能,比如你可以将你的 FLASH 划分为几个区 (就象硬盘的 C 盘和 D 盘一样), 每一个区作为不同的用途,在我的系统里我将我的 FLASH 分为了 4 个区。FLASH 分区的函数在/drivers/mtd/maps/子目录下,几个不同的文件对应几种不同目标系统的 FLASH 分区情况,我只是对 tqm8xxl.c 做了修改,然后在编译内核时选择了【CFI Flash device mapped on TQM8xxL】一项即可了。

关于 FLASH 和文件系统,在后面的实例分析中将会做详细的介绍。

4. 3. 3 FEC 以太网介绍

FEC 快速以太网的驱动程序位于/arch/ppc/8xx_io/fec.c,基本没有做修改,一个问题就是可能你的系统如果所用的 PHY 和驱动程序中的 PHY 有所不同,你就要做相应的修改,然后在内核中添加自己的选项。

5. 4 编译控制

4. 4. 1 内核编译

我想大家对内核编译的步骤都很清楚吧?嵌入 linux 的内核编译和普通的内核编译稍微有一点不同,因为用 U-BOOT 引导内核,在进行引导之前我们要对它进行一次包装,好在我们内核的编译控制把这个工作搞定了,具体情况请阅读 U-BOOT 下的 README,在这里我只是告诉你普通的内核编译命令:

```
#make bzImage
```

所生成的内核映象文件不可以用 U-BOOT 直接引导的,而要用:

```
#make pImage 或 #make uImage 生成的内核映象才可以直接引导。
```

下面详细介绍一下我的内核编译步骤。

- 进入你的内核源代码目录
#cd /opt/linux (对于你来说可能是其它目录)

- 内核配置

如果你用的是通用的目标系统比如 FADS860 板,你可以直接运行命令:

```
#make FADS860_config
```

```
#make oldconfig
```

```
#make dep
```

```
#make pImage 或者 #make uImage
```

然后生成内核映象文件位于/arch/ppc/mbxboot/下面的 pImage 或者 uImage。

如果你用的是自己开发的目标系统,那么自己对内核配置一把吧(本来这也是一件很有趣的事情),运行命令:

```
#make menuconfig
```

会出现一个对话框,挨个选项进行配置吧,下面是我的关键组件的配置:

【Platform support->】 平台支持

选择 Processor Type 为 8xx

选择 Machine Type 为 SUMMER860T (这是我自己在 config.in 中添加的)

选中 Math emulation

其它选项都不要选。

【Gernel setup-→】 通用配置

选择 etworking support 网络支持当然要的

选择 Sysctl support

选择 System V IPC

后面两个选项我还没有深入研究，不知不选是不是也可以？

其它的都不要选了。

【Memory Technology Devices(MTD)-→】 MTD 配置

选择 <*>Memory Technology Device(MTD) support 记着选择为“built-in”而不是“module”（前面的选项为“*”而不是“M”）

选择 <*>Mtd partitioning support MTD 分区支持

选择 <*>Caching block device access to MTD devices 支持MTD 作为 BLOCK 设备访问

进入【RAM/ROM/FLASH chip drivers-→】选项选择加载 FLASH 的驱动

选择 <*> Detect flash chips by Common Flash Interface (CFI) probe

自动探测 CFI 接口的 FLASH 设备

选择 <*> Support for AMD/Fujistu flash chips 选择对 AMD/Fujistu FLASH 设备的支持

选择 <*> Support for ROM chips in bus mapping 支持 ROM 设备总线 MAP

其它选项不选，退回到上一级。

进入【Mapping drivers for chip access-→】选项

选择 <*> CFI Flash device mapped on TQM8xxL 支持 TQM8xxL 的 FLASH 分区

其它选项都不用选中。

【Networking Options-→】 网络选项

本选项全部为 built-in 方式。

选择<*> Packet socket

选择<*> Kernel/User netlink socket

选择<*> Unix domain sockets

选择<*> TCP/IP networking

选择<*> IP: multicasting 支持组播

选择<*> IP: kernel level autoconfiguration

选择<*> IP: BOOTP support

选择<*> IP: RARP support

其它的都没有选。

【File systems-→】 文件系统

该目录下包含内核对各种文件系统的支持，可以根据需要选择想要的文件系统。

选择<*> Jouralling Flash File System (JFFS) support

选择<*> Jouralling Flash File System V2 (JFFS2) support

内核支持 JFFS 和 JFFS2 文件系统

选择<*> /proc file system support 用于查看系统的信息

进入【Network File Systems-→】 网络文件系统子选项
选择<*> NFS file system support NFS 网络文件系统支持
选择<*> Provide NFSv3 client support 支持 NFS 版本 3 客户端
选择<*> Root file system on NFS 根文件系统采用 NFS

退回上级菜单，其它的都没有选中。

【MPC8xx CPM Options-→】 MPC860T CPM 配置

选择<*> 860T FEC Ethernet 支持 FEC 快速以太网口
选择<*> Support LX970 PHY 选择 PHY 为 LX970（你的可能不一样）
选择<*> Use SMC1 for UART 用 SMC1 作为串口
选择<*> Use SMC1 for Console 用 SMC1 作为系统的 Console
SMC1 Rx BD 和 Tx BD 数目设为 4，每个 BD 大小为 32 字节
 (4) Number of the Rx Buffer for SMC1
 (32) Size of the Rx Buffer for SMC1
 (4) Number of the Tx Buffer for SMC1
 (32) Size of the Tx Buffer for SMC1
选择 MPC860T 数据 CACHE 工作方式为 Copy-Back
 <*> Copy-Back Data Cache (else Writethrough)
其它的配置可以根据目标系统的不同灵活配置。

其它的象 USB、SCSI、IDE、Sound 等和目标系统没关系的选项统统都要去掉。

配置基本完成，保存内核配置（最好保存到一个文件中，以便下次可以直接调入）。然后运行：

```
#make dep
```

```
#make pImage
```

编译完成，生成的内核映像文件在/arch/ppc/mbxboot/目录下 pImage，OK 了！

4. 4. 2 内核调试选项

事实上在本次嵌入 linux 的移植过程中并没有对 linux 的内核进行什么调试，对我们来说比较关心的就是我们目标系统的外设工作是否正常，内核的配置中有一些调试的选项，比如 MTD 的调试，在其选项下面选中：

```
<*> Debugging
```

```
<3> Debugging verbosity (0 = quiet, 3 = noisy)
```

则内核在运行时输出 MTD 初始化和工作时的所有调试信息，其它象 JFFS, JFFS2 也有相应的调试选项选择。在前期目标系统和内核还没有完全正常运行前最好将这些选项全部选中，以方便问题的查找，运行稳定后再关闭这些选项，提高内核的启动速度。

4. 5 内核的装载

内核编译完成之后下一步就是把内核下载到目标系统上或者直接用 BDM 仿真器将内核映像文件烧写到 FLASH 上。调试过程中我们可能经常要对内核进行重新编译和修改，每次将内核烧写到 FLASH 上无疑令人无法忍受，从网口 download 内核映像文件是调试时的最佳选择，本节重点叙述如何通过以太网口将下载内核映像。

首先，主机系统要做相关设置，这在我们初期的“主机系统”一节中已经做过介绍。在目标系统，U-BOOT 启动之后要做一些参数设置才可以从主机 download 内核映像文件，以我们的系统为例，目标系统进入 U-BOOT 的后按任意键，设置参数：

设置你主机的 IP 地址

```
=>setenv serverip 200.162.162.239
```

设置目标系统的 IP 地址

```
=>setenv ipaddr 200.162.162.190
```

设置网关 IP 地址

```
=>setenv gatewayip 200.162.162.1
```

设置子网掩码

```
=>setenv netmask 255.255.255.0
```

设置目标系统 FEC MAC 地址

```
=>setenv ethaddr 08:00:3e:26:9d:5b
```

设置目标系统名称

```
=>setenv hostname summer
```

然后保存参数

```
=>saveenv
```

运行 TFTPBOOT 命令，下载内核映像文件

```
=>tftpboot 0x100000 pImage      (0x100000 为下载内核后保存文件的起始 RAM 地址)
```

如下：

```
=>tftpboot 0x100000 pImage
```

```
ARP broadcast 1
```

```
TFTP from server 200.162.162.239; our IP address is 200.162.162.190
```

```
Filename 'pImage'.
```

```
Load address: 0x100000
```

```
Loading: #####
```

```
#####
```

```
done
```

```
Bytes transferred = 543429 (84ac5 hex)
```

内核映像下载完毕。

U-BOOT 提供了 FLASH 的擦写命令，你可以将 RAM 中的数据拷贝到 FLASH 中去（当然这里需要 FLASH 驱动的支持，实际上所用的命令都是 cp，但改命令会判断你的目标地址是哪里，如果是 FLASH，那么就会调用 FLASH 的写操作将数据写入 FLASH 中），我的系统中将内核映像拷贝到 FLASH 的偏移地址 0x20000，大小为 0xa0000，如下：

```
=>cp 0x100000 0x200000 0xa0000
```

将内核映像拷贝到了 FLASH 中。

4.6 引导嵌入 linux

下面就让我们的 linux 跑起来吧！

调试过程中，我选择 linux 的根文件系统为 NFS 网络文件系统，这样做的原因是因为目前我的 flash 太小，而且在开发上层应用程序时，采用 NFS 也比较灵活，在整个系统包括上层应用程序开发完毕之后在 FLASH 上建立一个 JFFS2 或者 CRAMFS 等文件系统。

目标系统要做如下设置：

```
=>setenv rootpath /opt/nfsfilesystem/
=>setenv bootargs root=/dev/nfs rw nfsroot=$(serverip):$(rootpath)
ip=$(ipaddr):$(serverip):$(gatewayip):$(netmask):$(hostname)::off
=>setenv bootcmd bootm 0x2820000 (从 FLASH 中启动内核)
或者
=>setenv bootcmd cp 0x2820000 0x100000 0xa0000\;bootm 0x100000
将内核映像先从 FLASH 拷贝到 RAM 然后从 RAM 启动内核
如果想每次从主机系统 download 内核然后启动，可以设为：
=>setenv bootcmd tftpboot 0x100000\;bootm 0x100000
然后保存参数设置
=>saveenv
重新启动系统
=>reset
OK!来看看吧！
=>tftpboot 0x100000 pImage-sc855t
ARP broadcast 1
TFTP from server 200.162.162.239; our IP address is 200.162.162.190
Filename 'pImage-sc855t'.
Load address: 0x100000
Loading: #####
#####
done
Bytes transferred = 577235 (8ced3 hex)
=>bootm 0x100000
## Booting image at 00100000 ...
   Image Name:   Linux-2.4.4
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    577171 Bytes = 563.6 kB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
   Uncompressing Kernel Image ... OK
Linux version 2.4.4 (root@linux-guo) (gcc version 2.95.3 20010111
(prerelease/fr
anzo/20010111)) #37 Thu Jun 26 15:43:46 CST 2003
On node 0 totalpages: 8192
zone(0): 8192 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel          command          line:          root=/dev/nfs          rw
nfsroot=200.162.162.239:/opt/nfsfilesystem/
ip=200.162.162.190:200.162.162.239:200.162.162.239:255.255.255.0:Summer
::off
Decrementer Frequency: 3125000
```



Warning: real time clock seems stuck!
Calibrating delay loop... 49.76 BogoMIPS
Memory: 30580k available (1120k kernel code, 424k data, 48k init, 0k highmem)
Dentry-cache hash table entries: 4096 (order: 3, 32768 bytes)
Buffer-cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 8192 (order: 3, 32768 bytes)
Inode-cache hash table entries: 2048 (order: 2, 16384 bytes)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Starting kswapd v1.8
CPM UART driver version 0.03
ttyS0 on SMC1 at 0x0280, BRG1
pty: 256 Unix98 ptys configured
block: queued sectors max/low 20224kB/6741kB, 64 slots per queue
eth0: FEC ENET Version 0.2, FEC irq 9, MII irq 12, addr 08:00:3e:26:9d:5b
devfs: v0.102 (20000622) Richard Gooch (rgooch@atnf.csiro.au)
devfs: boot_options: 0x2
JFFS version 1.0, (C) 1999, 2000 Axis Communications AB
JFFS2 version 2.1. (C) 2001 Red Hat, Inc., designed by Axis Communications AB.
Number of erase regions: 4
Primary Vendor Command Set: 0002 (AMD/Fujitsu Standard)
Primary Algorithm Table at 0040
Alternative Vendor Command Set: 0000 (None)
No Alternate Algorithm Table
Vcc Minimum: 2.7 V
Vcc Maximum: 3.6 V
No Vpp line
Typical byte/word write timeout: 16 祇
Maximum byte/word write timeout: 512 祇
Full buffer write not supported
Typical block erase timeout: 1024 祇
Maximum block erase timeout: 16384 祇
Chip erase not supported
Device size: 0x200000 bytes (2 MiB)
Flash Device Interface description: 0x0002
- supports x8 and x16 via BYTE# with asynchronous interface
Max. bytes in buffer write: 0x1
Number of Erase Block Regions: 4
Erase Region #0: BlockSize 0x4000 bytes, 1 blocks
Erase Region #1: BlockSize 0x2000 bytes, 2 blocks
Erase Region #2: BlockSize 0x8000 bytes, 1 blocks
Erase Region #3: BlockSize 0x10000 bytes, 31 blocks
Amd/Fujitsu Extended Query Table v1.0 at 0x0040

TQM8xxL Bank 0: JEDEC Device ID is 0xC4. Assuming broken CFI table.

TQM8xxL Bank 0: Swapping erase regions for broken CFI table.

number of JEDEC chips: 1

0: offset=0x0,size=0x10000,blocks=31

1: offset=0x1f0000,size=0x8000,blocks=1

2: offset=0x1f8000,size=0x2000,blocks=2

3: offset=0x1fc000,size=0x4000,blocks=1

TQM flash bank 0: Using static image partition definition

Creating 2 MTD partitions on "TQM8xxL Bank 0":

0x00000000-0x00020000 : "ppcboot"

0x00020000-0x00200000 : "jffs2"

NET4: Linux TCP/IP 1.0 for NET4.0

IP Protocols: ICMP, UDP, TCP, IGMP

IP: routing cache hash table of 512 buckets, 4Kbytes

TCP: Hash tables configured (established 2048 bind 2048)

NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.

Looking up port of RPC 100003/2 on 200.162.162.239

Looking up port of RPC 100005/2 on 200.162.162.239

VFS: Mounted root (nfs filesystem).

Freeing unused kernel memory: 48k init

INIT: version 2.78 booting

Welcome to Scom Linux 2.0 (Summer)

Press 'I' to enter interactive startup.

Mounting proc filesystem: [OK]

Configuring kernel parameters: [OK]

Activating swap partitions: [OK]

Setting hostname Summer: [OK]

INIT: Entering runlevel: 3

Entering non-interactive startup

Starting xinetd: [OK]

Scom Linux 2.0 (Summer)

Kernel 2.4.4 on an MPC8xx

Summer login:

启动完毕之后，敲个命令试一试：

#ls

没有问题!Linux 的 porting 过程也差不多了。

4. 7 关于文件系统

在嵌入 linux 移植过程中，文件系统是一个重要的环节，在本次嵌入 linux 的移植过程中，以本节做为整个移植工作的结尾。

4. 7. 1 简介

文件系统按物理位置可以分为本地文件系统 LFS(Local File System)和网络文件系统 NFS(Network File System)。本地文件系统的组件要求目标系统要有大的 FLASH 或者 DOC

(Disk On Chip) , 将在主机系统组织好的文件系统拷贝到目标系统上去, 一般的做法都是用 NFS 文件系统启动嵌入 Linux, 然后将组织好的文件系统拷贝到目标系统。NFS 文件系统对目标系统的记忆 MEMORY 没有多大的要求, 但需要主机系统的配合, 工作方式就是常见的服务器/客户机模型, 主机系统作为服务器, 目标系统为客户机。采用 NFS 文件系统要求内核要支持 BOOTP 或 DHCP, TFTP 和 NFS, 编译内核时要将这些功能选上去。

4. 7. 2 嵌入 Linux 文件系统分析

嵌入 linux 启动时需要一个根文件系统, 可以以多种方式组织这个根文件系统, 本次移植先后使用了 NFS 网络文件系统和 JFFS2 文件系统。嵌入 linux 的文件系统一般包含一下几个目录/bin, /sbin, /etc, /boot, /dev, /lib, /mnt, /proc, /usr。/bin 目录下包含常用的用户命令如 sh,bash,mount,cp,ls,ping 等命令; /sbin 目录包含所有系统命令如 reboot, shutdown 等; 配置文件和一些启动脚本位于/etc 目录下; 内核映像位于/boot 目录下; /dev 目录包含系统所有的设备文件; /lib 目录包含系统所需的库文件; /mnt 目录只是用于挂载, 可以是空目录; /proc 目录是/proc 文件系统的主目录, 所有关于系统启动和状态的信息都位于该目录下; /usr 目录包含用户选取的命令, 可以为空。

在建立文件系统时, 要根据自己的需要选取所要的命令和库, 如果你的目标系统的文件系统载体 (FLASH 或者 DOC) 不够大, 那就要好好裁减一下了, 下面是我的文件系统结构:

/bin 下的命令有:

bash, date, sh, login, mount, umount, cp, ls, ping

/sbin 下包含:

mingetty, reboot, halt, sulogin, init, fsck, telinit, mkfs.jffs2,mkfs

/etc 包含:

HOSTNAME, Bashrc, fstab, group, inittab, nsswitch, pam.d, passwd, pwdb.conf, rc.d, rc0.d, rc1.d, rc2.d, rc3.d, rc4.d, rc5.d, rc6.d, rc.local,init.d 等

/dev 至少要包含:

console, kmem, mem, tty1, ttyS0, mtd, mtd0, mtd1, mtd2, mtdblock, mtdblock0, mtdblock1, mtdblock2

/lib 包含:

libc.so.6: Linux 系统中所有命令的基本库文件

ld-linux.so.2: 基本库文件 libc.so.6 的装载程序库

libcom_err.so.2: 对应命令出错处理的装载程序库

libcrypt.so.2: 对应加密处理的程序库

libpam.so.0: 对应可拆卸身份验证模块的程序库

libpam_misc.so.2: 对应可拆卸身份验证模块解密用的程序库

libuuid.so.2: 对应身份识别信息的程序库

libtermcap.so.2: 对应描述终端和脚本的程序库

security: 此目录用来提供安全性所需的配置, 与 libpam.so.0 配合使用

4. 7. 2 DOC 支持

虽然我的目标系统中没有用到 DOC, 但本文仍然对其做简要介绍。

DOC(Disk On Chip)是 M-System 公司生产的一种大容量的闪存盘。采用 TRUEFFS 技术, 与平常的 FLASH 相比具有容量大, 价格便宜的有点, 如果你的目标系统需要大容量的掉电记忆体, DOC 是一个不错的选择, 详情请参阅 <http://www.m-sys.com/>。

4. 7. 3 实例分析—在 FLASH 上构造 JFFS2 文件系统

最后，我以一个实际的例子来叙述一下如何在 FLASH 构造一个 JFFS2 文件系统。

由于在我的目标系统 FLASH 太小，只有 2M，而且还要去掉将近 1M 空间用于存放 U-BOOT、内核映像，所以没有用建立的 JFFS2 文件系统作为根文件系统，只是介绍了在 FLASH 上制作一个文件系统的过程。

我把 FLASH 2M 的空间分为了 3 个区，分区的实现在 linux 根目录下/drivers/mtd/maps/目录下，我对 TQM8xxL (tqm8xxl.c) 的分区做了改动，作为我的分区，实际只要修改代码中的结构体 tqm8xxl_partitions[] 即可以：

```
static struct mtd_partition tqm8xxl_partitions[] = {
    {
        name:    "u-boot",
        offset:   0x00000000,
        size:     0x00020000,
    },

    {
        name:    "kernel",
        offset:   0x00020000,
        size:     0x000a0000,
    },

    {
        name:    "jffs2",
        offset:   0x000c0000,
        size:     0x00140000,
    }
}
```

很简单，第一行是你 partition 的名字，第二行是在 FLASH 中的偏移量，第三行是 partition 的大小。紧随其后的还有一个结构体：

static struct mtd_partition tqm8xxl_fs_partitions[]{} 是用于 linux2.4.x 以前版本的，在 linux2.4.x 版本中不用，我将它屏蔽掉了。

源文件中要改的就这么多，然后重新编译内核：

#make menuconfig

打开内核选项：

选择 <*>Memory Technology Device(MTD) support 记着选择为“built-in”而不是“module”（前面的选项为“*”而不是“M”）

选择 <*>Mtd partitioning support MTD 分区支持

选择 <*>Caching block device access to MTD devices 支持 MTD 作为 BLOCK 设备访问

进入【RAM/ROM/FLASH chip drivers->】选项选择加载 FLASH 的驱动

选择 <*> Detect flash chips by Common Flash Interface (CFI) probe

自动探测 CFI 接口的 FLASH 设备

选择 <*> Support for AMD/Fujistu flash chips 选择对 AMD/Fujistu FLASH 设备的支持

选择 <*> Support for ROM chips in bus mapping 支持 ROM 设备总线 MAP
其它选项不选，退回到上一级。

进入【Mapping drivers for chip access->】选项
选择 <*> CFI Flash device mapped on TQM8xxL 支持 TQM8xxL 的 FLASH 分区

其它选项都不用选中。

编译内核：

```
#make dep
```

```
#make pImage
```

生成内核映像文件在 arch/ppc/mbxboot/目录下，重新引导内核，进入 linux 系统，运行命令：

```
#more /proc/mtd 查看
```

```
dev:   size  erasesize  name
mtd0: 00020000 00010000 "ppcboot"
mtd1: 000a0000 00010000 "kernel"
mtd2: 00140000 00010000 "jffs2"
```

这里显示的 mtd0-2 是 FLASH 分区后模拟的三个设备，是字符型的设备，我们后面用的 mtdblock0-2 是对应三个 partitions 的 block 设备。

此时用的是 NFS 文件系统，退到根目录，运行：

```
#mkfs.jffs2 -d filesystem -o jffs2.image
```

制作 jffs2 文件系统，filesystem 是我制作的一个很小的包含几个文件的目录，生成的 jffs2 文件系统文件为 jffs2.image。在 /mnt/ 下建立目录 jffs2：

```
#mkdir /mnt/jffs2
```

将 jffs2 文件系统拷贝到 FLASH：

```
#cp jffs2.image /dev/mtdblock2
```

挂接 flash 设备到 /mnt/jffs2

```
#mount /dev/mtdblock2 /mnt/jffs2 -t jffs2
```

```
jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x001dc000: 0xe6a7 id
jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x001dc004: 0x626f id
jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x001dc008: 0x636d id
jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x001dc00c: 0x626f id
jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x001dc010: 0x6d20 id
jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x001dc014: 0x3030 id
jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x001dc018: 0x3000 id
jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x001dc01c: 0x7564 id
jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x001dc020: 0x7465 id
jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x001dc024: 0x3932 id
Further such events for this erase block will not be printed
JFFS2: Erase block at 0x001d0000 is not formatted. It will be erased
```

不知这些信息是否不正确，正在查询这个问题。

查看 jffs2 文件系统：

```
sh-2.05a# cd /mnt/jffs2
```

```
sh-2.05a# ls
```

```
app bin dev etc home lib mnt proc root sbin tmp usr var
```

```
sh-2.05a#
```

OK！制作的文件系统都在里面了，就这么简单。如果你将一个完整的文件系统拷贝到 FLASH，然后想用它作为根文件系统，可以在 U-BOOT 启动时设置启动参数：

#setenv bootargs root=/dev/mtdblock2 然后保存参数并重新引导 U-BOOT 即可以即可以。

对于制作 ROMFS 和 CramFS 文件系统，只需将制作文件系统映象时的命令更改一下即可以实现，如下：

```
#mkfs.cramfs -d filesystem -o cramfs.image
```

```
#mkfs.romfs -d filesystem -o romfs.image
```

然后拷贝到 FLASH 中即可以了。

关于在 FLASH 和 DOC 上实现 JFFS 请参考：

<ftp://ftp.uk.linux.org/pub/people/dwmw2/mtd/cvs/mtd/mtd-jffs-HOWTO.txt>

关于文件系统的详细介绍，请参阅：

<http://www.tldp.org/HOWTO/Filesystems-HOWTO-9.html>

关于 MTD 的详细信息，请参阅：

<http://www.linux-mtd.infradead.org/>

4.8 小结

至此，关于 U-BOOT 和嵌入 Linux 的移植工作全部结束了，后面的章节会介绍如何在嵌入 linux 进行应用程序开发以及目前比较流行的网络管理 SNMP 和 WEB 模块的实现。

在移植的过程中，最大的感触就是要多读多看，最好加入几个比较有人气的邮件组，象我在本次移植开始时就加入了 linuxppc 的邮件组 <http://lists.linuxppc.org/>，工作过程中有什么问题可以到这里面去查找一下相关的解释，如果查不到也可以发新的邮件，会有很多高手给你解答的，感谢那些默默无闻给予帮助的人，感谢 DENX 软件中心的！

本手册仅仅就我工作的过程做简单的叙述，涉及的知识面只是冰山一角，中间难免出现一些错误，希望能得到大家的指点。

===未完待续===